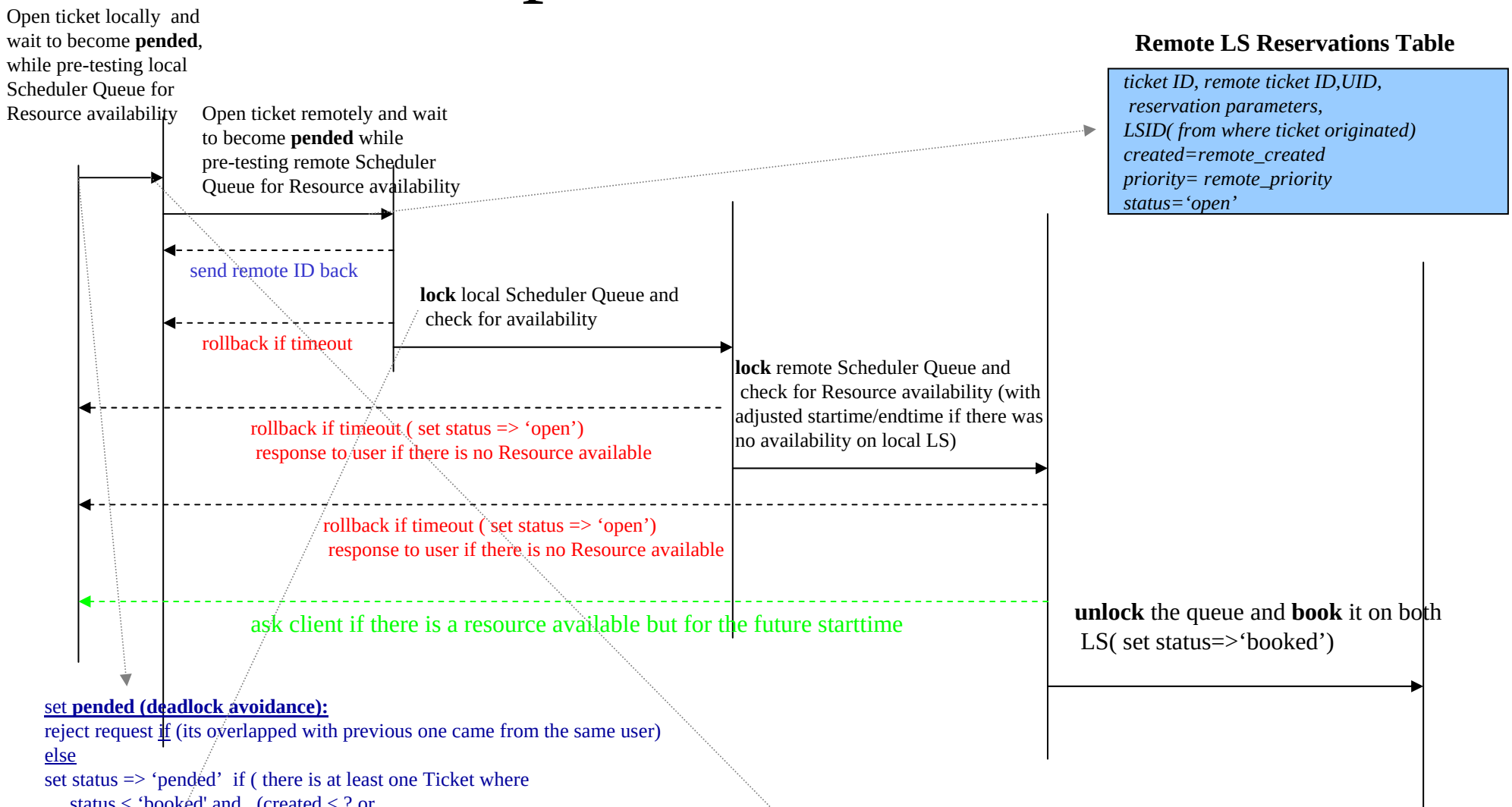
Lambda Station

Reservation Service Work Plan

Describe Data Model

- Client specifies src ip+p, dst ip+p, bandwidth, time slot, protocol (optional)
- Service reservation ticket (reservation ID)
- Ticket status (Boolean?)
- Flow specification (client params + additional info such as DSCP)
- Local site configuration. Needs to be standardized but not used by Web Service

Open Service Ticket



Remote LS Reservations Table

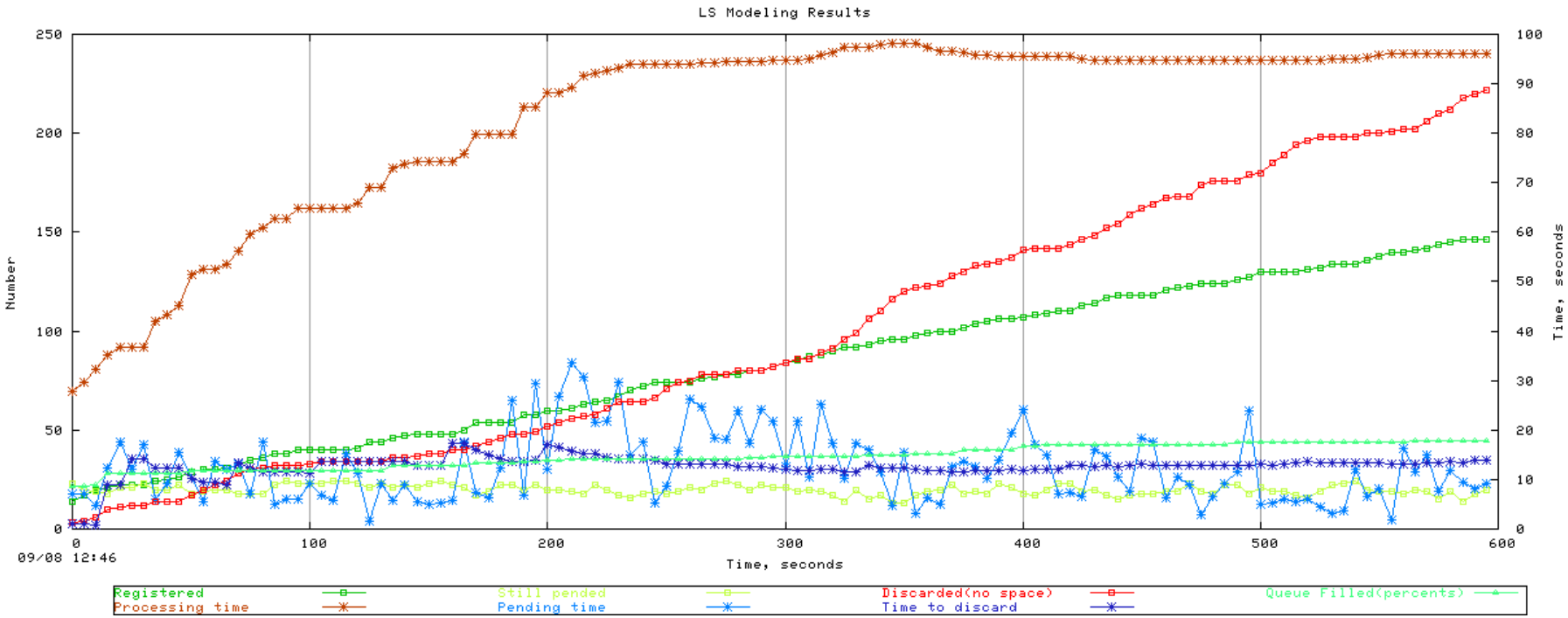
ticket ID, remote ticket ID,UID,
reservation parameters,
LSID(from where ticket originated)
created=remote_created
priority= remote_priority
status='open'

set **pended (deadlock avoidance):**
reject request if (its overlapped with previous one came from the same user)
else
set status => 'pended' if (there is at least one Ticket where
status < 'booked' and (created < ? or
(created = ? and ((inBW + outBW) > ? or
((inBW + outBW) = ? and (starttime < ? or priority > ?)))))

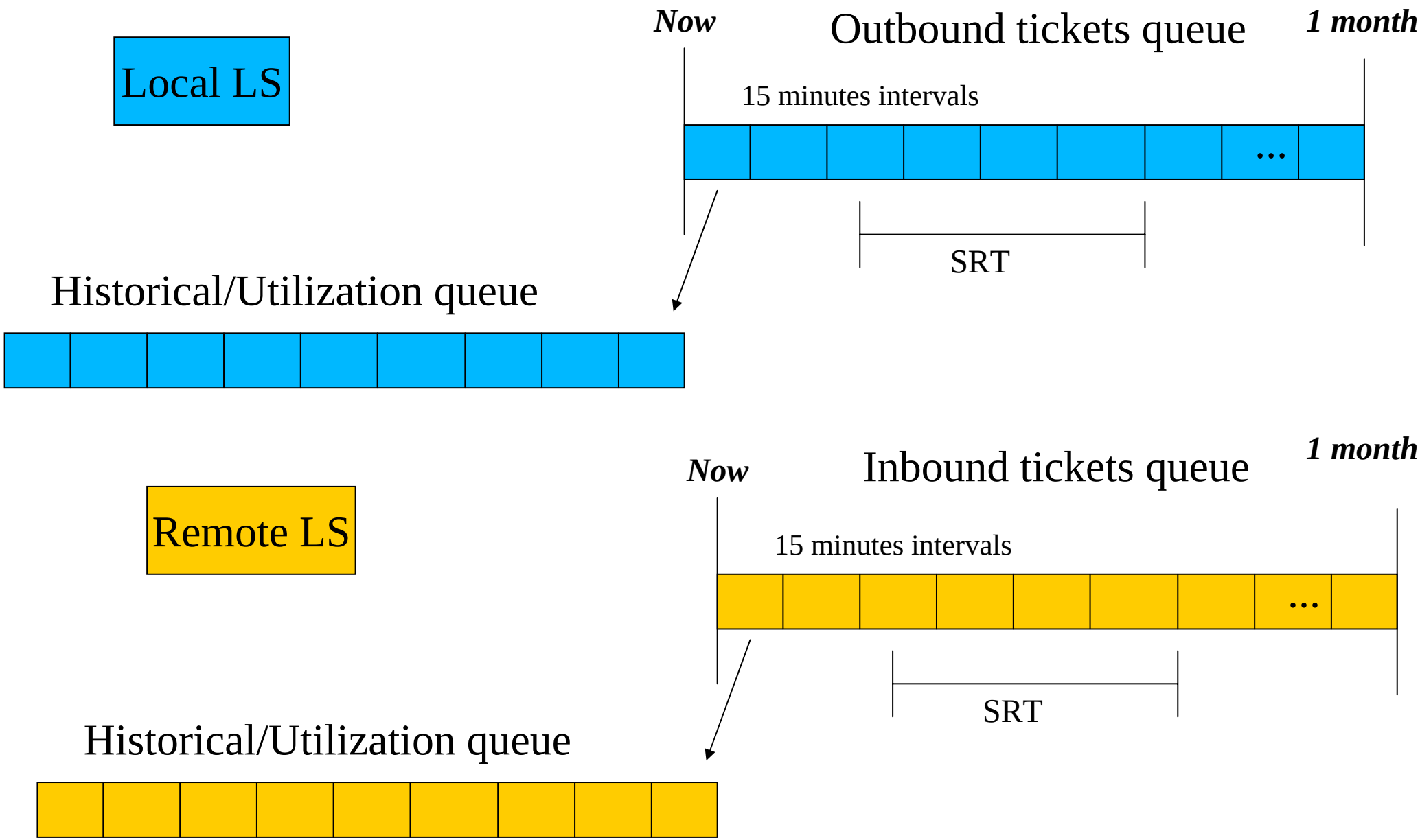
Local LS Reservations Table

ticket ID, UID
reservation parameters,
priority=#LS (every LS has some number, preset or generated by LS Discovery
status = 'open'
created = now() (up to second resolution due MySQL limitations,
could be improved to millisecond if switched to PostgreSQL)

locking Queue:
*Fine locking granularity could be achieved by utilizing transactional DB, such as InnoDB (MySQL) or Postgresql without AutoCommit and row locking if its not required then simple table locking would be sufficient (simulation with 4 LSs * 3 clients per station with ~5 sec connection delay showed ~100 sec per request to get booked), see graph on next page.*


LS Modeling Results
Number
Time, seconds
Time, seconds
09/08 12:46
0
50
100
150
200
250
0
10
20
30
40
50
60
70
80
90
100
0
100
200
300
400
500
600
Registered
Processing time
Still pended
Pending time
Discarded(no space)
Time to discard
Queue Filled(percents)

Scheduler Internals



Explanation

- mutex is done by locking on DB level (table locking or row locking)
- deadlock avoidance is implemented by prioritizing requests
- For every new **Outbound** Service Ticket (SRT check if SRT will fit (<= Available_BW) on Remote LS with **Inbound Tickets Queue (ITQ),** get response (**Yes** or **No** or **No, but reservation is available in the nearest future (< 1 day ?)**, then check if SRT will fit (<= Available_BW) in **Outbound Tickets Queue (OTQ)** time slot requested by user/application (U/A), get response, process both responses and give (U/A) choice to confirm or decline available reservation (giving choice is optional, we can start with **Yes** or **No)**. If user or user's application getting **No** then it would be helpful for user to create visual representation of ongoing reservation's table.
- If U/A accepts then add requested BW into corresponded intervals on ***Local LS Scheduler OTQ*** and ***Remote LS Scheduler ITQ*** and create new records in *Reservations* table on Local LS and *Reservations* table on Remote LS.
- Every 15 minutes move first interval into the right end of Outbound Historical/Utilization Queue on Local LS and do follow the algorithm described above on remote one(could be compared with real MRTG based utilization).
- Do the same just by reversing the names (Outbound -> Inbound) for **Inbound** Service Reservation tickets
- We will create OTQ/ITQ one pair per LinkID per LS for now, considered that we are not policing local resources
- Naming schema should be valid according to Local LS Network Topology XML Schema
- Special monitoring service will be showing up reservation table progress for local LS and resource availability

Identify Operations

- Identify the operations that can be performed on that data model
  - Start with the WSDL that was used for the Perl prototype
  - Apply the data model to the WSDL
  - Include client -> LS and LS -> LS operations
    - May split into two WSDLs later
  - Each LS controls the reservations for its own resources

Generate Code

- Generate java stubs from the WSDL
  - jclarens makes this trivial
- Fill out the simple operations to test the basic infrastructure
  - sayHello/echo method

Fill out Operations

- Implement the remainder of the WSDL operations
  - Generate Javadoc to describe operations and their implementation
  - Each operation must have a unit test to verify that it works

User Interface

- UI for making and viewing reservations is not a priority
  - Wait until after the basic server side functionality exists
  - UI should use RPC calls to get information
    - Clean separation of presentation and control layers
    - Both Javascript and JSP can do RPC